$1.80

Apple Assembly Line

Volume 8 -- Issue 1 November, 1987

Update on Drawing Circles

In case you missed it, Richard Miner presented a nice refinement to Dick Pountain's Circle Drawing Algorithm a in letter to the editor of Byte Magazine, December, 1987, pages 26-30. Miner's method allows you to use X- and Y-scale factors, so that you can cope with non-square aspect ratios on video screens and printers.

Furthermore, Brent Iverson has published an article on Hi-Res circle drawing in Nibble Magazine, January 1988, pages 68-71. He uses the same algorithm I did in my September article, and converts it to assembly language using MicroSparc's MacroSoft macros. The resulting code ($35A bytes) takes over three times as much memory the program I published in the September AAL, but it was probably easier to write.

Webster Said It

Do you know who Noah Webster was? His name is on practically every American dictionary, because he wrote the first one. (I have a copy of his small 1806 edition and another of his very large one from 1828.) Called America's foremost pioneer lexicographer, he mastered 20 languages including Hebrew and Greek. In 1833 he published his own revision of the King James Version of the Bible. I bought a reprint of it this week (Baker Book House, 1987), and intend to read it through in the coming year. Webster said, "The Bible is the chief moral cause of all that is good, and the best corrector of all that is evil, in human society; the best book for regulating the temporal concerns of men, and the only book that can serve as an infallible guide to future felicity."

Still a Bug in IIgs Smartport

Alan J.Silver reports in the Jan 88 issue of Open Apple that the new version 01 IIgs ROMs clobber locations $57 thru $5A on the caller's Direct Page when you make a Smartport call to the firmware in slot 5. The older ROMs clobbered the same locations in "true" page zero, as reported in AAL, May, 1987, page 26.

(Pretty) Fast DOS Textwriter.....................R.R. Bukrey

Bill Morgan's article (Feb 87) on writing very large DOS text files very fast was interesting to me. Not too long ago, I had tackled a similar problem. I had modified Cornelis Bongers' Cross Assembler (Micro on the Apple, Vol. 3) to produce output compatible with the DOS Toolkit Assembler, and I needed a way to get the output to disk. On discovering DOS's aversion to handling text files from machine language, I realized I had to write something from scratch. The result was TEXTFILE.

TEXTFILE is fast (5.5 sectors per second), although presumably not as fast as Bill's program. As noted in Bill's article, the speed advantage comes mainly from keeping the VTOC, T/S list and catalog sectors all in memory, rather than reading and writing them repeatedly to and from disk.

Thus, while TEXTFILE is not super-fast, I think it does offer some advantages: it doesn't require the file space to be previously allocated. It reads the disk directory and either locates the desired file or creates it. The file size produced by TEXTFILE is limited only by available disk space.

Finally, TEXTFILE requires no patches to DOS, and since it was written with no space restrictions, it is self-contained, needing no BASIC caller to help it along. Probably the main advantage of the program, in retrospect, is that it offers a pretty straightforward tutorial on DOS file management.

We can outline the workings of TEXTFILE by taking a quick tour through the listing. To make the listing a little shorter, I have turned off the listing of the macro expansion. The >SET macro, defined in lines 1510-1560, is used to store an buffer address into a pointer.

Right off, we note that we haven't been careful to avoid Applesoft's turf. Lines 1050-1210 use HIMEM and other important BASIC pointers. These would probably have to be relocated if we were intending to link with a running BASIC program.

Lines 1620-1850 input the file name and check its syntax.

The SCAN routine (line 1890) looks thru the directory for the specified name. A matchup skips ahead to FOUND (line 2940). While scanning, it saves the sector number of the first deleted file that happens by. On no matchup, this entry will be used for creating the new file. If no deleted entries are handy, the first blank one is used. Lacking even that, it quits via 'disk full error.'

If a new file entry is to be created, this is done in lines 2690-2930.

With the directory set up, we can move on to the VTOC. Lines 3040-3160 read it into its buffer and initialize counters and pointers. One important point: the first byte of the VTOC is used as a change flag: if the VTOC never changes, we can skip writing it back to the disk when we are done.

```
S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners...................$20
ProDOS Upgrade Kit for Version 2.0 DOS owners........................$30
Source Code of S-C Macro 2.0 (DOS or ProDOS)........each, additional $100
S-C DisAssembler (ProDOS only)...without source code $30, with source $50
RAK-Ware DISASM (DOS only).......without source code $30, with source $50
ProVIEW (ProDOS only)................................................$20
Full Screen Editor for S-C Macro (with complete source code).........$49
S-C Cross Reference Utility......without source code $20, with source $50
S-C Word Processor (with complete source code).......................$50
DP18 and DPFP, including complete source and object code.............$50
ES-CAPE (Extended S-C Applesoft Program Editor)..........
                          Including Version 2.0 With Source Code.........$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)....$30
Bag of Tricks 2 (Quality Software)........................($49.95)  $45 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
   (Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
                8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)
Beagle Bros. Applesoft Compiler (ProDOS only)..............($74.95)  $65 *
Copy II Plus (Central Point Software)......................($39.95)  $30 *

AAL Quarterly Disks..........................each $15, or any four for $45
 Each disk contains the source code from three issues of AAL:
   Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100 *
Envelopes for Diskette Mailers...........................  6 cents each

Sider 20 Meg Hard Disk, includes controller & software........($695) $550 +
Sider 40 Meg Hard Disk, includes controller & software........($995) $860 +

Minuteman 250 Uninterruptible Power Supply....................($359) $320 +
Minuteman 300 Uninterruptible Power Supply....................($549) $490 +

65802 Microprocessor (Western Design Center).......................  $25 *
quikLoader EPROM System (SCRG)................................($179) $170 *
PROmGRAMER (SCRG)..........................................($149.50) $140 *

"Exploring the Apple IIgs", Gary B. Little.................($22.95)  $21 *
"Apple IIgs Technical Reference", Fischer..................($19.95)  $19 *
"65816/65802 Assembly Language Programming", Fischer........($21.95)  $20 *
"Programming the 65816", Eyes & Lichty.....................($22.95)  $21 *
"Apple //e Reference Manual", Apple Computer...............($24.95)  $23 *
"Apple //c Reference Manual", Apple Computer...............($24.95)  $23 *
"ProDOS-8 Technical Reference Manual", Apple Computer.......($29.95)  $27 *
"ProDOS-16 Technical Reference Manual", Apple Computer......($29.95)  $27 *
"Apple IIgs Firmware Reference", Apple Computer............($24.95)  $23 *
"Apple IIgs Hardware Reference", Apple Computer............($24.95)  $23 *
"ProDOS Inside and Out", Doms & Weishaar...................($16.95)  $16 *
"DOStalk Scrapbook", Weishaar & Kersey.....................($14.95)  $14 *
"Beneath Apple ProDOS", Worth & Lechner....................($19.95)  $18 *
"Beneath Apple DOS", Worth & Lechner.......................($19.95)  $18 *
"Inside the Apple //c", Little.............................($19.95)  $18 *
"Inside the Apple //e", Little.............................($19.95)  $18 *
"Understanding the Apple //e", Sather......................($24.95)  $23 *
"Understanding the Apple II", Sather.......................($22.95)  $21 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($18.95)  $18 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95)  $18 *
"Enhancing Your Apple II, vol. 1", Lancaster...............($15.95)  $15 *
"Enhancing Your Apple II, vol. 2", Lancaster...............($17.95)  $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20 *
"Microcomputer Graphics", Myers............................($14.95)  $14 *
"Assembly Lines -- the Book", Wagner.......................($19.95)  $12 *

            * On these items add $2.00 for the first item and
              $.75 for each additional item for US shipping.
            + Inquire for shipping cost.
              Foreign customers inquire for postage needed.
         Texas residents please add 8% sales tax to all orders.
```

Next, the track/sector list is either read (old file) or created (new file). In the latter case, the subroutine GETFREE (lines 4610-5060) scans the VTOC for the next free sector, starting in track $22 and working down through track $03. Only one pass through the VTOC is made, and no attempt is made to mimic DOS's 'optimization,' i.e., starting near the catalog track and looking in both directions for empty space.

Once the T/S list is in place, we can start dumping data to the disk (lines 3540-4100). The loop at line 3590 fills the data buffer with 16 copies of a 16-character string. This is where the user would insert his/her own data-generating routine. As might be expected, any existing T/S list is used until exhausted, then extended by GETFREEing as needed.

This demo version of TEXTFILE is set up to quit when the sector counter reaches zero (having been initialized to 50). Any real application would similarly have to indicate end of data.

With the data safely sequestered on disk, we can restore the directory, T/S list and VTOC to disk (lines 4270-4570), and call it a day.

A parting thought: the following Applesoft program demonstrates the speed advantage of TEXTFILE over BASIC: a factor of five. Enjoy.

```
100 D$=CHR$(4)
110 PRINT D$"OPEN TTT" : PRINT D$"WRITE TTT"
120 FOR I = 1 TO 50
130   FOR J = 1 TO 16
        : PRINT "THIS IS A TEST."
        : NEXT
140 NEXT I
150 PRINT D$"CLOSE"
```

```
                  SAVE FAST.TEXT.SAVE
                  1010 *--------------------------------
                  1020 *   FAST TEXTFILE SAVE PROGRAM, BY R. R. BUKREY
                  1030 *--------------------------------
 42-              1050 A4L     .EQ $42
 48-              1060 STATUS  .EQ $48      P-REGISTER
 71-              1070 NAMLEN  .EQ $71      FILE NAME LENGTH
 72-              1080 PTR     .EQ $72      BUFFER POINTER & SECTOR COUNTER
 74-              1090 DIRSEC  .EQ $74      DIRECTORY SECTOR USED
 75-              1100 FILPTR  .EQ $75      POINTER TO FILE ENTRY IN DIR BUFFR
 77-              1110 TST     .EQ $77      TRK OF T/S LIST
 78-              1120 TSS     .EQ $78      SECT OF T/S LIST
 79-              1130 TYP     .EQ $79      FILE TYPE
 7A-              1140 NTSL    .EQ $7A      NO. OF T/S LISTS
 7B-              1150 VY      .EQ $7B      OFFSET INTO VTOC
 7C-              1160 VB1     .EQ $7C      2-BYTE BUFFER FOR VTOC ROL
 7D-              1170 VB2     .EQ $7D
 7E-              1180 VTTRK   .EQ $7E      LAST TRACK ALLOCATED
 7F-              1190 TSPTR   .EQ $7F      T/S POINTER
 80-              1200 EOD     .EQ $80      END OF DATA FLAG
 81-              1210 CTR     .EQ $81
                  1220 *--------------------------------
 0200-            1230 KBUFF   .EQ $200
                  1240 *--------------------------------
```

```
03D3-              COLDOS .EQ $3D3
03D9-              RWTS   .EQ $3D9
03E3-              GETIOB .EQ $3E3
                   *---Data Areas inside DOS--------
9600-              BUFF   .EQ $9600           DATA BUFFER
9700-              TSB    .EQ $9700           T/S LIST BUFFER
B3BB-              VTOC   .EQ $B3BB           VTOC BUFFER
B4BB-              DBUFF  .EQ $B4BB           DIRECTORY BUFFER
                   *
B7E8-              IOB    .EQ $B7E8           I/O CONTROL BLOCK
B7E9-              SLOT   .EQ IOB+1
B7EA-              DRIVE  .EQ IOB+2
B7EB-              VOL    .EQ IOB+3           0=ANY
B7EC-              TRK    .EQ IOB+4
B7ED-              SECT   .EQ IOB+5
B7F0-              BUFFAD .EQ IOB+8
B7F4-              OPER   .EQ IOB+12          1=READ 2=WRITE
B7F5-              RETCOD .EQ IOB+13
                   *---Subroutines inside DOS-------
A702-              DOSERR .EQ $A702           PRINT ERROR MSG
B7D6-              ZBUFF  .EQ $B7D6           ZERO BUFFER POINTED TO BY A4
                   *---Subroutines in Monitor-------
FD8E-              CROUT  .EQ $FD8E
FC58-              HOME   .EQ $FC58
FD6F-              GETLIN .EQ $FD6F
                   *--------------------------------
                          .MA SET      >SET VARIABLE,VALUE
                          LDA #]2
                          STA ]1
                          LDA /]2
                          STA ]1+1
                          .EM
                   *--------------------------------
                          .OR $803
                   *--------------------------------
                   *   GET FILE NAME
                   *--------------------------------
                   TEXTFILE
0803- 20 58 FC            JSR HOME
0806- 20 8E FD            JSR CROUT
0809- 20 8E FD            JSR CROUT
080C- 20 6F FD            JSR GETLIN
080F- 8A                  TXA
0810- F0 21               BEQ .2              ZERO LENGTH
0812- E0 1F               CPX #$1F
0814- B0 1D               BCS .2              NAME TOO LONG
0816- 86 71               STX NAMLEN          SAVE LENGTH
0818- AD 00 02            LDA KBUFF           1ST CHAR A LETTER?
081B- C9 C1               CMP #$C1
081D- 30 14               BMI .2
081F- C9 DB               CMP #$DB
0821- 10 10               BPL .2
0823- A0 01               LDY #1
0825- B9 00 02     .1     LDA KBUFF,Y
0828- C9 8D               CMP #$8D            CR = END OF NAME
082A- F0 0A               BEQ SCAN
082C- C9 AC               CMP #$AC            NO COMMAS ALLOWED
082E- F0 03               BEQ .2
0830- C8                  INY
0831- D0 F2               BNE .1              ALWAYS
0833- 4C 1F 0B     .2     JMP SYNERR
                   *--------------------------------
                   *   SCAN DIRECTORY FOR NAME
                   *--------------------------------
0836- A9 60        SCAN   LDA #$60     USE SLOT 6
0838- 8D E9 B7            STA SLOT
083B- A2 00               LDX #0
083D- 8E EB B7            STX VOL      USE ANY VOLUME #
0840- 86 74               STX DIRSEC
                   *   DIRSEC IS ZERO UNTIL A DELETED ENTRY OCCURS.
                   *   THEN IT HOLDS THE SECTOR OF THAT ENTRY.
                   *   FINALLY, IT IS THE SECTOR OF THE ENTRY ACTUALLY USED.
0842- E8                  INX
0843- 8E EA B7            STX DRIVE
0846- 8E F4 B7            STX OPER
0849- A9 0F               LDA #$0F     START WITH SECTOR $0F
084B- 8D ED B7            STA SECT
084E- 20 C1 0A            JSR DIRIOB          SET IOB BUFF ADDR & TRK
0851- 85 73               STA PTR+1
```

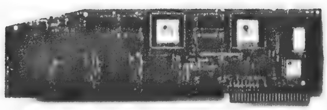
```
                   2040 *---Read next directory sector---
0853- 20 02 0B 2050 .1      JSR R.W
0856- AD F0 B7 2060         LDA BUFFAD
0859- 18       2070         CLC
085A- 69 0B    2080         ADC #$0B
               2090 *---Point to next filename-------
085C- 85 72    2100 .2      STA PTR
085E- A0 00    2110         LDY #0
0860- B1 72    2120         LDA (PTR),Y       1ST CHAR OF FILE ENTRY
0862- F0 64    2130         BEQ BLANK         BLANK ENTRY
0864- C9 FF    2140         CMP #$FF          DELETED FILE?
0866- F0 26    2150         BEQ .6
0868- B1 72    2160 .3      LDA (PTR),Y
086A- 99 77 00 2170         STA TST,Y         SAVE T/S & TYPE
086D- C8       2180         INY
086E- C0 03    2190         CPY #3
0870- D0 F6    2200         BNE .3
0872- A2 00    2210         LDX #0
0874- B1 72    2220 .4      LDA (PTR),Y       COMPARE NAME IN FILE ENTRY
0876- DD 00 02 2230         CMP KBUFF,X       WITH INPUT NAME
0879- D0 1A    2240         BNE .7            QUIT IF NO MATCH
087B- E8       2250         INX
087C- C8       2260         INY
087D- E4 71    2270         CPX NAMLEN        DONE WITH INPUT NAME?
087F- 90 F3    2280         BCC .4            NO, GO DO REST
0881- C0 21    2290 .5      CPY #$21          30 CHARS MAX + 3
0883- F0 77    2300         BEQ FOUND
0885- B1 72    2310         LDA (PTR),Y       MAKE SURE REST OF ENTRY IS BLANK
0887- C9 A0    2320         CMP #$A0
0889- D0 0A    2330         BNE .7
088B- C8       2340         INY
088C- D0 F3    2350         BNE .5            ALWAYS
088E- A5 74    2360 .6      LDA DIRSEC
0890- D0 03    2370         BNE .7
0892- 20 BA 08 2380         JSR SAVDIR        SAVE POINTERS TO 1ST DELETED ENTRY
0895- A5 72    2390 .7      LDA PTR
0897- 18       2400         CLC
0898- 69 23    2410         ADC #$23          BUMP POINTER TO NEXT ENTRY
089A- C9 0C    2420         CMP #$0C
089C- D0 02    2430         BNE .8
089E- E6 73    2440         INC PTR+1         PAGE CROSSED
08A0- C9 BB    2450 .8      CMP #$BB
08A2- D0 B8    2460         BNE .2            GO READ NEXT ENTRY
               2470 *---Next directory sector--------
08A4- CE ED B7 2480         DEC SECT          NEXT SECTOR
08A7- F0 04    2490         BEQ .9
08A9- C6 73    2500         DEC PTR+1
08AB- D0 A6    2510         BNE .1            ALWAYS
               2520 *  HAVE NOW READ ALL DIR SECTS W/NO MATCH, NO BLANK ENTRIES
08AD- A5 74    2530 .9      LDA DIRSEC        ANY DELETED ENTRIES?
08AF- D0 03    2540         BNE .10           YES, GO USE ONE
08B1- 4C 19 0B 2550         JMP FULL          NO, DIRECTORY FULL, SO QUIT
08B4- EE ED B7 2560 .10     INC SECT          RESET TO SECTOR 1
08B7- 4C CC 08 2570         JMP FE1
               2580 *--------------------------------
08BA- AD ED B7 2590 SAVDIR LDA SECT           SAVE DIRECTORY POINTERS
08BD- 85 74    2600         STA DIRSEC
08BF- A5 72    2610         LDA PTR
08C1- 85 75    2620         STA FILPTR
08C3- A5 73    2630         LDA PTR+1
08C5- 85 76    2640         STA FILPTR+1
08C7- 60       2650         RTS
               2660 *--------------------------------
               2670 *   USE FILE ENTRY FOUND, OR BUILD NEW ONE
               2680 *--------------------------------
08C8- A5 74    2690 BLANK   LDA DIRSEC        USE DELETED ENTRY, IF ANY
08CA- F0 0D    2700         BEQ FE2           NONE, GO USE BLANK ONE
08CC- CD ED B7 2710 FE1     CMP SECT          FIND DELETED ENTRY
08CF- F0 0B    2720         BEQ FE3           IN CURRENT SECTOR, GO USE IT
08D1- 8D ED B7 2730         STA SECT          NOT HERE. GO BACK & GET IT
08D4- 20 02 0B 2740         JSR R.W
08D7- 90 03    2750         BCC FE3           ALWAYS
08D9- 20 BA 08 2760 FE2     JSR SAVDIR        USE CURRENT SECT
08DC- A0 03    2770 FE3     LDY #3            MOVE NAME TO ENTRY
08DE- A2 00    2780         LDX #0
08E0- BD 00 02 2790 FE4     LDA KBUFF,X
08E3- 91 75    2800         STA (FILPTR),Y
```

```
08E5- C8        2810        INY
08E6- E8        2820        INX
08E7- E4 71     2830        CPX NAMLEN
08E9- 90 F5     2840        BCC FE4
08EB- C0 21     2850 FE5    CPY #$21           30 CHARS MAX + 3
08ED- F0 07     2860        BEQ FE6            DONE
08EF- A9 A0     2870        LDA #$A0           BLANK REST OF NAME FIELD
08F1- 91 75     2880        STA (FILPTR),Y
08F3- C8        2890        INY
08F4- D0 F5     2900        BNE FE5            ALWAYS
08F6- A9 FF     2910 FE6    LDA #$FF
08F8- 85 79     2920        STA TYP            RAISE NEW ENTRY FLAG
08FA- 30 11     2930        BMI RVT            ALWAYS
08FC- A5 79     2940 FOUND  LDA TYP            CHECK FILE TYPE
08FE- F0 0A     2950        BEQ FE8            UNLOCKED TEXT FILE, USE IT
0900- C9 80     2960        CMP #$80           LOCKED TEXT FILE?
0902- F0 03     2970        BEQ FE7
0904- 4C 22 0B  2980        JMP TYPERR         WRONG TYPE
0907- 4C 1C 0B  2990 FE7    JMP LOCK
090A- 20 BA 08  3000 FE8    JSR SAVDIR         SAVE FILE ENTRY POINTERS
                3010 *--------------------------------
                3020 *   READ VTOC AND INIT COUNTERS & POINTERS
                3030 *--------------------------------
090D- 20 D1 0A  3040 RVT    JSR VTIOB          SET IOB BUFF ADDR & SECT
0910- 20 02 0B  3050        JSR R.W
0913- A2 00     3060        LDX #0             INITIALIZE...
0915- 8E BB B3  3070        STX VTOC           VTOC CHANGE FLAG,
0918- 86 72     3080        STX PTR            FILE SECTOR COUNTER,
091A- 86 73     3090        STX PTR+1
091C- 86 80     3100        STX EOD            END OF DATA FLAG.
091E- A9 C2     3110        LDA #$C2
0920- 85 7B     3120        STA VY             INDEX FOR READING VTOC
0922- A9 22     3130        LDA #$22
0924- 85 7E     3140        STA VTTRK          CURRENT TRACK FOR VTOC
0926- AD 33 0B  3150        LDA NSEC           NO OF SECTORS IN TESTFILE
0929- 85 81     3160        STA CTR
                3170 *--------------------------------
                3180 *   READ T/S LIST INTO TSB, OR BUILD NEW ONE
                3190 *--------------------------------
092B- CA        3200        DEX
092C- 86 7A     3210        STX NTSL           SET UP T/S LIST COUNTER
                3220 *   NTSL WILL BE ONE LESS THAN NO OF SECTORS USED FOR T/S LIST
092E- A5 79     3230        LDA TYP            NEW OR OLD FILE?
0930- D0 17     3240        BNE TS2            NEW, GO BUILD T/S LIST
0932- A5 77     3250        LDA TST            OLD, READ T/S LIST
0934- 8D EC B7  3260        STA TRK
0937- A5 78     3270        LDA TSS
0939- 8D ED B7  3280        STA SECT
093C- 20 E1 0A  3290 TS1    JSR TSIOB          SET UP IOB BUFF ADDR
093F- A9 01     3300        LDA #1
0941- 8D F4 B7  3310        STA OPER           READ
0944- 20 02 0B  3320        JSR R.W
0947- 90 1E     3330        BCC TS5            ALWAYS
0949- 20 6A 0A  3340 TS2    JSR GETFREE        SECT FOR T/S LIST
094C-           3350 TS3    >SET A4L,TSB       BUILD T/S LIST IN TSB
0954- 20 D6 B7  3360        JSR ZBUFF          CLEAR IT FIRST (RETURNS Y=0)
0957- A5 7A     3370        LDA NTSL
0959- 10 0C     3380        BPL TS5            NEXT LINES ONLY ONCE
095B- 84 79     3390        STY TYP
095D- B9 77 00  3400 TS4    LDA TST,Y
0960- 91 75     3410        STA (FILPTR),Y      SET TYPE & T/S FOR NEW FILE
0962- C8        3420        INY
0963- C0 03     3430        CPY #3
0965- D0 F6     3440        BNE TS4
0967- A0 03     3450 TS5    LDY #3
0969- 20 07 0A  3460        JSR SAVTS          LABEL T/S LIST W/ITS OWN DISK LOCN
096C- 20 14 0A  3470        JSR INCPTR         BUMP FILE SECTOR COUNTER
096F- E6 7A     3480        INC NTSL           AND T/S LIST COUNTER
0971- A9 0C     3490        LDA #$0C
0973- 85 7F     3500        STA TSPTR          INIT T/S POINTER
                3510 *--------------------------------
                3520 *   WRITE A DATA SECTOR FROM BUFF
                3530 *--------------------------------
0975-           3540 WD1    >SET A4L,BUFF
097D- 20 D6 B7  3550        JSR ZBUFF          CLEAR DATA BUFFER (RETURNS Y=0)
0980- A2 10     3560        LDX #$10           TEST MSG REPEATED 16X PER SECTOR
0982- 86 79     3570        STX TYP
0984- CA        3580        DEX
```

```
0985- BD 34 0B 3590 WD0    LDA TEXT,X        FILL BUFFER WITH TEXT
0988- 99 00 96 3600        STA BUFF,Y
098B- C8       3610        INY
098C- CA       3620        DEX
098D- 10 F6    3630        BPL WD0
098F- C6 79    3640        DEC TYP
0991- F0 04    3650        BEQ WD2           BUFFER FULL
0993- A2 0F    3660        LDX #$0F
0995- 10 EE    3670        BPL WD0           ALWAYS
0997- C6 81    3680 WD2    DEC CTR
0999- D0 02    3690        BNE WD3
099B- E6 80    3700        INC EOD           NO MORE DATA COMING
099D- A4 7F    3710 WD3    LDY TSPTR
099F- B9 00 97 3720        LDA TSB,Y         GET NEXT T/S PAIR
09A2- F0 0F    3730        BEQ WD4           NONE. GO FIND A FREE SECTOR
09A4- 8D EC B7 3740        STA TRK           GOT IT. SAVE TRACK...
09A7- C8       3750        INY
09A8- B9 00 97 3760        LDA TSB,Y
09AB- 8D ED B7 3770        STA SECT          AND SECTOR
09AE- C8       3780        INY          UPDATE T/S POINTER
09AF- 84 7F    3790        STY TSPTR         AND SAVE IT, TOO.
09B1- D0 14    3800        BNE WD5           ALWAYS
09B3- 20 6A 0A 3810 WD4    JSR GETFREE       SECTOR FOR DATA
09B6- A4 7F    3820        LDY TSPTR
09B8- 20 07 0A 3830        JSR SAVTS         PUT T & S IN T/S LIST
09BB- 84 7F    3840        STY TSPTR
09BD- A5 77    3850        LDA TST           AND IN IOB ALSO
09BF- 8D EC B7 3860        STA TRK
09C2- A5 78    3870        LDA TSS
09C4- 8D ED B7 3880        STA SECT
09C7-          3890 WD5    >SET BUFFAD,BUFF
09D1- A9 02    3900        LDA #2
09D3- 8D F4 B7 3910        STA OPER          WRITE
09D6- 20 02 0B 3920        JSR R.W           DATA SECTOR TO DISK
09D9- 20 14 0A 3930        JSR INCPTR        BUMP FILE SECTOR COUNTER
09DC- A5 80    3940        LDA EOD           END OF DATA?
09DE- D0 3B    3950        BNE RC1           YES, GO RESTORE CATALOG
09E0- A5 7F    3960        LDA TSPTR         END OF T/S LIST?
09E2- D0 91    3970        BNE WD1           NO, GO GET MORE DATA
09E4- AD 01 97 3980        LDA TSB+1         YES, CHECK FOR NEXT LIST
09E7- F0 0C    3990        BEQ WD6           NONE. GO BUILD ONE
09E9- 8D EC B7 4000        STA TRK           SAVE T/S OF NEXT LIST...
09EC- AD 02 97 4010        LDA TSB+2
09EF- 8D ED B7 4020        STA SECT
09F2- 4C 3C 09 4030        JMP TS1           THEN GO READ IT.
09F5- 20 6A 0A 4040 WD6    JSR GETFREE       SECTOR FOR NEW T/S LIST
09F8- A0 01    4050        LDY #1
09FA- 20 07 0A 4060        JSR SAVTS         SAVE LINKS IN CURRENT T/S LIST
09FD- 88       4070        DEY
09FE- 8C F4 B7 4080        STY OPER          WRITE
0A01- 20 EC 0A 4090        JSR SAVTSB        CURRENT T/S LIST TO DISK
0A04- 4C 4C 09 4100        JMP TS3           GO BUILD NEXT T/S LIST
               4110 *--------------------------------
0A07- A5 77    4120 SAVTS  LDA TST
0A09- 99 00 97 4130        STA TSB,Y
0A0C- C8       4140        INY
0A0D- A5 78    4150        LDA TSS
0A0F- 99 00 97 4160        STA TSB,Y
0A12- C8       4170        INY
0A13- 60       4180        RTS
               4190 *--------------------------------
0A14- E6 72    4200 INCPTR INC PTR
0A16- D0 02    4210        BNE .1
0A18- E6 73    4220        INC PTR+1
0A1A- 60       4230 .1     RTS
               4240 *--------------------------------
               4250 *   RESTORE CATALOG SECTORS TO DISK
               4260 *--------------------------------
0A1B- A0 21    4270 RC1    LDY #$21          MOVE FILE LENGTH TO CATALOG ENTRY
0A1D- A5 72    4280        LDA PTR
0A1F- 91 75    4290        STA (FILPTR),Y
0A21- C8       4300        INY
0A22- A5 73    4310        LDA PTR+1
0A24- 91 75    4320        STA (FILPTR),Y
0A26- 20 C1 0A 4330        JSR DIRIOB        SETUP BUFF ADDR & TRK
0A29- A5 74    4340        LDA DIRSEC
```

```
0A2B- 8D ED B7 4350          STA SECT
0A2E- A9 02    4360          LDA #2
0A30- 8D F4 B7 4370          STA OPER              WRITE
0A33- 20 02 0B 4380          JSR R.W               DIRECTORY SECTOR TO DISK
0A36- A4 7F    4390          LDY TSPTR             CLEAR REST OF T/S BUFFER
0A38- F0 08    4400          BEQ RC3
0A3A- A9 00    4410          LDA #0
0A3C- 99 00 97 4420 RC2      STA TSB,Y
0A3F- C8       4430          INY
0A40- D0 FA    4440          BNE RC2
0A42- 20 EC 0A 4450 RC3      JSR SAVTSB            SAVE T/S LIST TO DISK
0A45- AD BB B3 4460          LDA VTOC              VTOC CHANGE FLAG
0A48- F0 16    4470          BEQ RC4               SKIP VTOC IF UNCHANGED
0A4A- A9 00    4480          LDA #0
0A4C- 8D BB B3 4490          STA VTOC              CLEAR CHANGE FLAG
0A4F-          4500          >SET BUFFAD,VTOC
0A59- A9 11    4510          LDA #$11              TRACK $11, SECTOR 0
0A5B- A0 00    4520          LDY #0
0A5D- 20 FC 0A 4530          JSR CALL.RWTS.AY      WRITE VTOC TO DISK
0A60- A5 80    4540 RC4      LDA EOD
0A62- D0 03    4550          BNE EXIT
0A64- 4C 19 0B 4560          JMP FULL              DISK FULL ERROR IF NOT END OF DATA
0A67- 4C D3 03 4570 EXIT     JMP COLDOS            EXIT TO BASIC
               4580 *--------------------------------
               4590 *   ROUTINE TO SCAN VTOC FOR NEXT FREE SECTOR
               4600 *--------------------------------
0A6A- A4 7B    4610 GETFREE LDY VY
0A6C- 88       4620 V1       DEY
0A6D- B9 BB B3 4630          LDA VTOC,Y
0A70- 18       4640          CLC
0A71- 88       4650          DEY
0A72- 79 BB B3 4660          ADC VTOC,Y            TRACK FULL?
0A75- D0 11    4670          BNE V2                NO, GO FIND FREE SECTOR
0A77- B0 0F    4680          BCS V2                THEY COULD ADD TO ZERO!
0A79- 88       4690          DEY              YES, TRY NEXT ONE
0A7A- 88       4700          DEY
0A7B- C6 7E    4710          DEC VTTRK
0A7D- C0 42    4720          CPY #$42              DON'T LOOK BELOW TRK 3
0A7F- D0 EB    4730          BNE V1
0A81- A9 00    4740          LDA #0
0A83- 85 80    4750          STA EOD               CLR FLAG TO FORCE DISK FULL ERROR
0A85- 4C 1B 0A 4760          JMP RC1               EXIT AFTER RESTORING CATALOG SECTORS
0A88- B9 BB B3 4770 V2       LDA VTOC,Y            MOVE BIT MAP TO ROL BUFFER
0A8B- 85 7C    4780          STA VB1
0A8D- C8       4790          INY
0A8E- B9 BB B3 4800          LDA VTOC,Y
0A91- 85 7D    4810          STA VB2
0A93- C8       4820          INY
0A94- 84 7B    4830          STY VY                SAVE Y FOR NEXT TIME
0A96- A2 0F    4840          LDX #$0F              SECTOR
0A98- A9 80    4850 V5       LDA #$80              MASK BIT
0A9A- 18       4860          CLC
0A9B- 26 7D    4870 V3       ROL VB2
0A9D- 26 7C    4880          ROL VB1
0A9F- B0 06    4890          BCS V4                FREE SECTOR FOUND
0AA1- CA       4900          DEX              NOT  FOUND, TRY NEXT ONE
0AA2- 4A       4910          LSR
0AA3- 90 F6    4920          BCC V3
0AA5- B0 F1    4930          BCS V5                ALWAYS
0AA7- 86 78    4940 V4       STX TSS               SAVE SECTOR
0AA9- 88       4950          DEY
0AAA- E0 08    4960          CPX #8                USE 2ND MAP BYTE?
0AAC- 90 01    4970          BCC V6                NO, USE 1ST
0AAE- 88       4980          DEY
0AAF- 49 FF    4990 V6       EOR #$FF              COMPLEMENT AC
0AB1- 39 BB B3 5000          AND VTOC,Y            CLEAR BIT = SECTOR USED
0AB4- 99 BB B3 5010          STA VTOC,Y            UPDATE VTOC
0AB7- A9 01    5020          LDA #1
0AB9- 8D BB B3 5030          STA VTOC              SET CHANGE FLAG
0ABC- A5 7E    5040          LDA VTTRK
0ABE- 85 77    5050          STA TST               SAVE TRACK
0AC0- 60       5060          RTS
               5070 *--------------------------------
               5080 *   IOB SET-UPS USED MORE THAN ONCE
               5090 *--------------------------------
```

```
0AC1- A9 11     5100 DIRIOB LDA #$11
0AC3- 8D EC B7  5110        STA TRK
0AC6-           5120        >SET BUFFAD,DBUFF
0AD0- 60        5130        RTS
0AD1-           5140 VTIOB  >SET BUFFAD,VTOC
0ADB- A9 00     5150        LDA #0
0ADD- 8D ED B7  5160        STA SECT
0AE0- 60        5170        RTS
0AE1-           5180 TSIOB  >SET BUFFAD,TSB
0AEB- 60        5190        RTS
                5200 *--------------------------------
                5210 *   ROUTINE TO SAVE T/S LIST TO DISK
                5220 *--------------------------------
0AEC-           5230 SAVTSB >SET BUFFAD,TSB
0AF6- AD 03 97  5240        LDA TSB+3     TRACK
0AF9- AC 04 97  5250        LDY TSB+4     SECTOR
                5260 *      JMP CALL.RWTS.AY    *** FALL INTO IT ***
                5270 *--------------------------------
                5280 *   RWTS CALLER.  EXITS THRU DOS IF ERROR OCCURS.
                5290 *--------------------------------
                5300 CALL.RWTS.AY
0AFC- 8D EC B7  5310        STA TRK
0AFF- 8C ED B7  5320        STY SECT
0B02- 20 E3 03  5330 R.W    JSR GETIOB
0B05- 20 D9 03  5340        JSR RWTS
0B08- A9 04     5350        LDA #4        IRQ OFF, DECIMAL OFF
0B0A- 85 48     5360        STA STATUS
0B0C- B0 01     5370        BCS .1        R/W ERROR
0B0E- 60        5380        RTS
0B0F- AE F5 B7  5390 .1     LDX RETCOD
0B12- E0 10     5400        CPX #$10
0B14- F0 0E     5410        BEQ ANYERR    $04 = WRITE PROTECT ERROR
0B16- 0A        5420        ASL           $08 = I/O ERROR
0B17- D0 0B     5430        BNE ANYERR    ...ALWAYS
                5440 *--------------------------------
0B19- A9 09     5450 FULL   LDA #9
0B1B- 2C        5460        .HS 2C
0B1C- A9 0A     5470 LOCK   LDA #$0A
0B1E- 2C        5480        .HS 2C
0B1F- A9 0B     5490 SYNERR LDA #$0B
0B21- 2C        5500        .HS 2C
0B22- A9 0D     5510 TYPERR LDA #$0D
0B24- 48        5520 ANYERR PHA
0B25- 20 8E FD  5530        JSR CROUT
0B28- 20 8E FD  5540        JSR CROUT
0B2B- 68        5550        PLA
0B2C- AA        5560        TAX
0B2D- 20 02 A7  5570        JSR DOSERR
0B30- 4C D3 03  5580        JMP COLDOS         EXIT TO BASIC
                5590 *--------------------------------
                5600 *   Dummy data for demonstration
                5610 *--------------------------------
0B33- 32        5620 NSEC   .HS 32        number of sectors in demo file
0B34- 8D        5630 TEXT   .HS 8D
0B35- AE D4 D3
0B38- C5 D4 A0
0B3B- C1 A0 D3
0B3E- C9 A0 D3
0B41- C9 C8 D4  5640        .AS -/.TSET A SI SIHT/
                5650 *--------------------------------
```



PC Transporter taps into the world's largest software library. Now your Apple can run most of the IBM software you use at work. And it opens a new world of communications programs, games and bulletin boards.

A universal disk drive controller.

PC Transporter supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes. You'll shift instantly between Apple ProDOS and IBM MS-DOS.

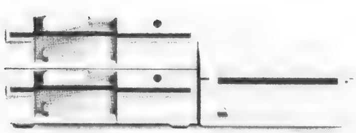
You'll need our versatile 5.25" 360K drive system to run IBM applications from 5.25" floppy disks. Use your Apple 5.25" drive for Apple 5.25" disks.

An Apple Disk 3.5 Drive will support the new 3.5"disks whether they're IBM MS-DOS formatted or Apple ProDOS formatted. The PC Transporter acts like an Apple Disk 3.5 Drive disk controller for IIGS, IIe, and II Plus users.

PC Transporter supports up to 5 drives in a number of combinations.

For example, you can connect a 5.25 Applied Engineering 360K dual-drive system directly to the card. Then plug two daisy-chained Apple 3.5 Drives (not the Apple UniDisk 3.5) to the dual-drive system. For a fifth drive, use a ProDOS file as an IBM hard disk.

PC Transporter controls Apple and IBM compatible disk drives. It supports 3.5" and 5.25" MS-DOS and ProDOS formatted diskettes.



Versatile data storage.

PC Transporter reads MS-DOS and translates it into Apple native ProDOS. You can store IBM programs and data on any ProDOS storage device including the Apple 3.5 Drive, Apple UniDisk™ 3.5, Apple 5.25" drive, SCSI or ProDOS compatible hard drives. (You can use the Apple UniDisk 3.5 with its own controller card for storing programs and data, but not for directly booting an IBM formatted disk.)

You can even use our 360K PC compatible drive for ProDOS storage and a 143K Apple 5.25" drive for MS-DOS storage.

Make your Apple speak IBM.

PC Transporter memory choices.

RAM in Apple mode:	RAM in IBM mode:	Price:
384K	**256K**	**$489.00**
512K	**384K**	**529.00**
640K	**512K**	**569.00**
768K	**640K**	**609.00**

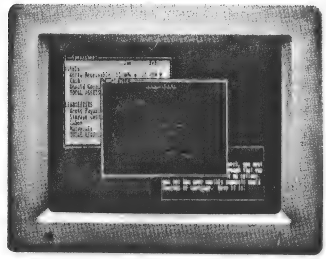
Note: The IBM mode is 128K less because the PC Transporter uses 128K for system memory.

IIGS Installation Kit	**49.00**
IIe/II Plus Installation Kit	**39.00**

PC Transporter Accessories

5.25" IBM Format 360K Drive Systems	
Single-Drive System	**269.00**
Dual-Drive System	**399.00**
Half-Height Drive (add to Single-Drive system to make Dual-Drive)	**135.00**
IBM-Style Keyboard (required for Apple II Plus. Requires IBM Keyboard Cable.)	**139.00**
IBM Keyboard Cable	**34.00**
Sony RGB Monitor	**499.00**
Analog RGB Cable (for use with Sony monitor)	**39.00**
Digital RGB Cable (for use with Sony monitor)	**39.00**
Digital RGB Adapter	**24.00**
ColorSwitch (included with IIGS Installation Kit)	**44.00**
128K ZIP PC Transporter Memory Expansion Chip Set	**40.00 per set**
8087-2 Math Co-processor Chip	**229.00**
Heavy Duty Power Supply (IIe and II Plus only)	**69.00**

See your dealer or call or send check or money order to Applied Engineering. MasterCard, VISA and COD welcome. Texas residents add 6¼% sales tax.



PC Transporter produces better IBM graphics than IBM. Analog is sharper than digital. So with an analog RGB monitor, PC Transporter's CGA graphics and text are superior to IBM's digital display — even while running IBM software!

And, you can also use an Apple composite monitor in IBM text or graphics mode.

Created by Apple's original designers.

The brains behind PC Transporter were also behind your Apple II.

The PC Transporter design team includes the former project managers for the creation of the Apple IIe and IIc. The co-designer of the Apple II disk controller. And the first full-time Apple programmer and author of the ProDOS operating system.

So you know the PC Transporter and your Apple were made for each other.

Support and service from the leader in Apple add-ons.

Applied Engineering sells more Apple peripheral boards than anyone else — including Apple Computer. So you know we'll be around after the sale.

PC Transporter comes with a 15-day money back guarantee. If you're not fully satisfied after using it, return it for a full refund. PC Transporter also comes with a 1-year warranty.

How to get your PC Transporter today.

See your dealer. Or call Applied Engineering any day between 9 a.m. and 11 p.m. CST at 214-241-6060.

Æ Applied Engineering

The Apple enhancement experts.

P.O. Box 798, Carrollton, TX 75006
214-241-6060
A Division of AE Research Corporation

A Strange Decimal-to-Binary Conversion....Bob Sander-Cederlof

Back in the 1950's I worked for a few years with the Bendix G-15D Computer. This machine was the ultimate personal computer of its day. The operator console consisted of an IBM Executive typewriter, with a few added switches. Mass storage was supplied by paper tape, both in loose coils and in cassettes (roughly the physical size of our present day VHS video cassettes). You got 2176 words of RAM, each with 29 bits, on a rotating magnetic drum. Let's see...that is less than 8K bytes. The three two-word registers and one one-word register also resided on the magnetic drum. The hardware instruction set included multiply and divide, and also some sophisticated logical field extraction operations. Speed? Well, it was plenty fast enough for its day. The basic unit, as described above, cost $50,000. In those days that was a very good price for a real computer, and engineering groups all over the country bought them with alacrity. You could also add a magnetic tape unit, a Calcomp X-Y plotter, a Digital Differential Analyzer, and more.

Believe it or not, during the entire lifetime of the product, which was over ten years, nobody ever wrote an assembler for the G-15. You had to program it either in raw hex, in a decimalized translation of the raw hex, or in an interpretive language. (We did eventually get the equivalent of a mini-assembler, with the auspicious name of "Altran".) The various intepretive languages supplied floating point math and simplified I/O, but it still looked like raw machine language. Everything was done with numbers, you could not use symbolic names for opcodes or operands.

There was one significant exception. In the early 60's a group of geniuses created a version of Algol for this machine. The compiler consisted of eight magazines full of paper tape! In case you never heard of Algol, you can think of it as the predecessor of Pascal.

In the middle 60's Control Data Corporation bought out the computer division of Bendix, and a few West Coast salesmen got the bright idea that these old beasts could get a second life in high schools and Junior Colleges. Part of my job at that time was to train high school teachers in using the G-15 and programming with one of the interpreters. Some of you may remember the name of Bob Albrecht, from the late 70's, the early days of Dr. Dobbs; he was also quite active in this project of setting up high schools with G-15s.

Well, anyway, you could do a lot with just a little back in those days. I stumbled over a pile of old G-15 manuals a few weeks ago, and out popped this fascinating decimal-to-binary conversion subroutine. I decided it was worth the effort to translate it into 6502 code. It converts a string of seven decimal digits in packed BCD form (or eight if you select the option in line 1050) to a 32-bit (29 in the G-15) binary value.

In my program I simulate the three two-word registers with four-byte variables named PLIER, CAND, and PROD. It is not as

many bits (32 bersus 58), but this program only needs 32 bits in each register. The code for the conversion is shown below in lines 1680-1960. When you realize that the EXTRACT and MULTIPLY subroutines I call here were simple machine language instructions in the G-15, you can see that the program was very compact in that machine. The EXTRACT subroutine simulates the G-15 instruction, which uses a binary mask to produce two results at once. The PROD (product) register is ANDed with the mask and the result stored in the CAND (multiplicand) register. After that, everywhere there are 1-bits in the mask the corresponding bits are cleared in the PROD register. For example, start with PROD = $12345678 and MASK = $0F0F0FFF. Afterwards CAND = $02040678 and PROD = $10305000.

The G-15 multiply instruction was unique, in that it could be told how many bits to multiply. My subroutine simulates that property by using the X-register to specify how many times to loop around, once for each bit. MULTIPLY adds the CAND*PLIER partial products to the PROD register.

There a a few secrets hidden in the value assembled at FACTOR. To simplify and speed up my MULTIPLY subroutine, FACTOR contains the 1's complement of the actual factor. The actual factor for eight digits is $AAC9F400. This is used in pieces: four bits = $A, three bits = $5, six bits = $19, and nine bits = $7D. Note that $A is 10, $5 is 10/2, $19 is 250 or 100/4, and $7D is 125 or 1000/8. Is it starting to make sense now?

If you look at the four masks, you will notice that the F's correspond to BCD digit positions. Think of the digit positions as D7 through D0, left to right. MASK0 causes digit D7 to multiplied by ten; MASK1 causes digits D7, D6, D4, and D1 to be multiplied by ten; MASK2 causes digits D7, D6, D5, and D2 to be multiplied by 100; and MASK3 causes digits D7 through D3 to be multiplied by 1000. The result is the same as D7*10^7 + D6*10^6 + ... + D1*10 + D0.

```
D7:  10*10*100*1000 = 10^7
D6:     10*100*1000 = 10^6
D5:        100*1000 = 10^5
D4:     10    *1000 = 10^4
D3:           *1000 = 10^3
D2:       *100      = 10^2
D1:    *10          = 10^1
D0:   untouched     = 10^0
```

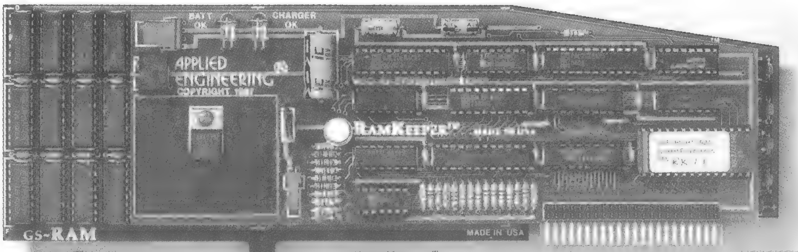
I hope I haven't lost you. If I have, please go back and read it again. I think it is really worth the effort! The idea of using an unfinished multiply simply MUST have other applications....

My demonstration program starts in line 1130. It allows you to type in a decimal number, and then prints the converted value in hex. Lines 1260-1570 read your input line and pack up the digits as BCD in the PROD register. Lines 1590-1660 print the four bytes of PROD in hex.

```
                  1000 *SAVE FUNNY.CONVERT.1
                  1010 *-------------------------------
                  1020 *    CONVERT 7- OR 8-DIGIT PACKED BCD VALUE
                  1030 *       TO BINARY
01-               1040 EIGHT  .EQ 1          =1 FOR 8 DIGITS, =0 FOR 7 DIGITS
                  1060 *-------------------------------
FD67-             1070 MON.RDLINE .EQ $FD67
FDDA-             1080 MON.PRBYTE .EQ $FDDA
FDED-             1090 MON.COUT   .EQ $FDED
0200-             1100 INBUF      .EQ $200
33-               1110 MON.PROMPT .EQ $33
                  1120 *-------------------------------
                  1130 T
0800- 20 1C 08    1140 .1      JSR GET.BCD.VALUE
0803- 90 16       1150         BCC .2         FINISHED
0805- 20 56 08    1160         JSR DISPLAY.PROD
0808- A9 BD       1170         LDA #"="
080A- 20 ED FD    1180         JSR MON.COUT
080D- A9 A4       1190         LDA #"$"
080F- 20 ED FD    1200         JSR MON.COUT
0812- 20 64 08    1210         JSR FUNNY.CONVERSION
0815- 20 56 08    1220         JSR DISPLAY.PROD
0818- 4C 00 08    1230         JMP .1
081B- 60          1240 .2      RTS
                  1250 *-------------------------------
                  1260 GET.BCD.VALUE
081C- A9 BD       1270         LDA #"="
081E- 85 33       1280         STA MON.PROMPT
0820- 20 67 FD    1290         JSR MON.RDLINE
0823- E0 01       1300         CPX #1         SEE IF EMPTY LINE
0825- 90 2E       1310         BCC .4         ...YES
0827- A2 04       1320         LDX #4         CLEAR PROD FIRST
0829- A9 00       1330         LDA #0
082B- 9D E0 08    1340 .1      STA PROD-1,X
082E- CA          1350         DEX
082F- D0 FA       1360         BNE .1
                  1370 *---ACCUMULATE NUMBER------------
0831- BD 00 02    1380 .2      LDA INBUF,X
0834- 49 B0       1390         EOR #"0"
0836- C9 0A       1400         CMP #10
0838- B0 1B       1410         BCS .4
083A- 0A          1420         ASL            POSITION IN HIGH NYBBLE
083B- 0A          1430         ASL
083C- 0A          1440         ASL
083D- 0A          1450         ASL
083E- A0 03       1460         LDY #3
0840- 2A          1470 .3      ROL
0841- 2E E4 08    1480         ROL PROD+3
0844- 2E E3 08    1490         ROL PROD+2
0847- 2E E2 08    1500         ROL PROD+1
084A- 2E E1 08    1510         ROL PROD
084D- 88          1520         DEY
084E- 10 F0       1530         BPL .3
0850- E8          1540         INX
0851- E0 08       1550         CPX #8
0853- 90 DC       1560         BCC .2
0855- 60          1570 .4      RTS
                  1580 *-------------------------------
                  1590 DISPLAY.PROD
0856- A0 00       1600         LDY #0
0858- B9 E1 08    1610 .2      LDA PROD,Y
085B- 20 DA FD    1620         JSR MON.PRBYTE
085E- C8          1630         INY
085F- C0 04       1640         CPY #4
0861- 90 F5       1650         BCC .2
0863- 60          1660         RTS
                  1670 *-------------------------------
                  1680 FUNNY.CONVERSION
0864- A0 02       1690         LDY #2         ONLY NEED 3 BYTES OF FACTOR
0866- B9 F5 08    1700 .1      LDA FACTOR,Y
0869- 99 D9 08    1710         STA PLIER,Y
086C- 88          1720         DEY
086D- 10 F7       1730         BPL .1
                  1740 *-------------------------------
                  1750   .DO EIGHT
086F- A2 03       1760         LDX #MASK0
0871- 20 97 08    1770         JSR EXTRACT
```

RamKeeper™

For the "Instant On" Apple IIGS.



GS-RAM™ RamKeeper™

Permanent Storage with an "Electronic Hard Disk"

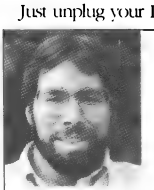
Now when you turn on your IIGS your favorite program can appear on screen in just a few seconds! With RamKeeper, your IIGS memory card will retain stored programs and stored data while your IIGS is turned off. RamKeeper allows you to divide your IIGS memory into part "electronic hard disk" and part RAM for your programs workspace—in almost any way you want and at anytime you want. GS-RAM, GS-RAM Plus, Apple IIGS memory card and most other IIGS memory cards are compatible with RamKeeper.

Supports Up to Two IIGS Memory Cards at the Same Time

If you bought your IIGS with Apple's memory card and later wished you had the GS-RAM, no problem. RamKeeper will support both cards plugged into RamKeeper simultaneously!

How it Works

Just unplug your IIGS memory card from your computer, plug your IIGS memory card into RamKeeper, plug RamKeeper into the IIGS memory slot. If you have another IIGS memory card, an additional card socket on RamKeeper will accommodate your second card. That's all there is to it!



"I've purchased several Applied Engineering products over the years. They're always well made and performed as advertised. I recommend them whole-heartedly."

Steve Wozniak, the creator of Apple Computer

Reliability from the Most Experienced

Applied Engineering has the most experience in the industry with battery-backed memory for the Apple so you are assured of the most reliable memory back-up system available. And in the world of battery-backed memory, reliability is everything! That's why Applied Engineering uses the more dependable Gel-Cell's instead of Ni-Cad batteries (if Ni-Cad's aren't discharged periodically, they lose much of their capacity). RamKeeper has close to 6 times (about 6 hours) the "total power failure" back-up time of other systems. When power returns, RamKeeper automatically re-charges the battery to a full charge. With power from your wall outlet, RamKeeper will back-up your IIGS memory cards RAM indefinately.

RamKeeper Has and Does It All!

- Allows instant access to your programs without slow disk delays
- Configure Kilobytes or Megabytes of instant ROM storage for your favorite programs
- Reduces power strain to your internal IIGS' power supply
- Contains back-up status L.E.D.'s
- Can support up to two IIGS memory cards simultaneously
- Supports both 256K installed memory chip boards like GS-RAM and the Apple IIGS Memory Expansion Card as well as 1 MEG installed memory chip boards like GS-RAM Plus
- 5-year hassle-free warranty
- 15 day money back guarantee
- Proudly made in the U.S.A.
- RamKeeper comes *complete* with battery, software and documentation

Only $179.00!

(GS-RAM card shown in photo not included)

Order Your RamKeeper Today!

See your dealer or call (214) 241-6060, 9:00-11:00 CST, 7 days a week, or send check or money order to Applied Engineering. MasterCard, VISA and C.O.D. welcome. Texas residents add 7% sales tax. Add $10.00 if outside U.S.A.

AE APPLIED ENGINEERING™

The Apple enhancement experts.

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006

Prices subject to change without notice

```
0874- A2 04              LDX #4        MULTIPLY 4 CYCLES
0876- 20 AD 08           JSR MULTIPLY
                    .FIN
               *--------------------------------
0879- A2 07              LDX #MASK1
087B- 20 97 08           JSR EXTRACT
087E- A2 03              LDX #3        MULTIPLY 3 CYCLES
0880- 20 AD 08           JSR MULTIPLY
               *--------------------------------
0883- A2 0B              LDX #MASK2
0885- 20 97 08           JSR EXTRACT
0888- A2 06              LDX #6        MULTIPLY 6 CYCLES
088A- 20 AD 08           JSR MULTIPLY
               *--------------------------------
088D- A2 0F              LDX #MASK3
088F- 20 97 08           JSR EXTRACT
0892- A2 09              LDX #9        MULTIPLY 9 CYCLES
0894- 4C AD 08           JMP MULTIPLY
               *--------------------------------
               EXTRACT
0897- A0 03              LDY #3
0899- B9 E1 08 .1        LDA PROD,Y
089C- 3D E5 08           AND MASKS,X
089F- 99 DD 08           STA CAND,Y
08A2- 59 E1 08           EOR PROD,Y
08A5- 99 E1 08           STA PROD,Y
08A8- CA                 DEX
08A9- 88                 DEY
08AA- 10 ED              BPL .1
08AC- 60                 RTS
               *--------------------------------
               MULTIPLY
08AD- 4E DD 08 .1        LSR CAND      MSBYTE
08B0- 6E DE 08           ROR CAND+1
08B3- 6E DF 08           ROR CAND+2
08B6- 6E E0 08           ROR CAND+3    LSBYTE
08B9- 0E DC 08           ASL PLIER+3   LSBYTE
08BC- 2E DB 08           ROL PLIER+2
08BF- 2E DA 08           ROL PLIER+1
08C2- 2E D9 08           ROL PLIER     MSBYTE
08C5- B0 0E              BCS .3        ...DO NOT ADD 'CAND
08C7- A0 03              LDY #3
08C9- B9 E1 08 .2        LDA PROD,Y
08CC- 79 DD 08           ADC CAND,Y
08CF- 99 E1 08           STA PROD,Y
08D2- 88                 DEY
08D3- 10 F4              BPL .2
08D5- CA       .3        DEX
08D6- D0 D5              BNE .1
08D8- 60                 RTS
               *--------------------------------
08D9-          PLIER    .BS 4          HI-BYTE FIRST
08DD-          CAND     .BS 4
08E1-          PROD     .BS 4
               *--------------------------------
               MASKS
                    .DO EIGHT
03-            MASK0    .EQ *-MASKS+3
08E5- F0 00 00
08E8- 00                .HS F0.00.00.00
                    .FIN
07-            MASK1    .EQ *-MASKS+3
08E9- FF 0F 00
08EC- F0                .HS FF.0F.00.F0
0B-            MASK2    .EQ *-MASKS+3
08ED- FF F0 0F
08F0- 00                .HS FF.F0.0F.00
0F-            MASK3    .EQ *-MASKS+3
08F1- FF FF F0
08F4- 00                .HS FF.FF.F0.00
               *--------------------------------
                    .DO EIGHT
08F5- 55 36 0B
08F8- FF       FACTOR .HS 55.36.0B.FF    10, 10, 100, 1000
                    .ELSE
               FACTOR .HS 53.60.BF.FF    10, 100, 1000
                    .FIN
               *--------------------------------
```

Getting a Pointer from a Handle..........Bob Sander-Cederlof

Did you read "Let's Get a Handle on this Memory", by Ken Kashmarek in the October 1987 Call APPLE, pages 61-63? Ken ably discusses what "Handles" and "Pointers" are in the Apple IIgs world, and gives some subroutines to use for finding data pointed to by them.

Handles and Pointers are part of a hierarchy of addresses that enable you to find things the Memory Manager and others have hidden and moved around in RAM. For example, the Memory Manager gives you a Handle, which is a 24-bit address pointing to a Master Pointer, which is in turn a 24-bit address pointing to your Memory block. The Memory manager is free to move the actual memory block around, as long as it keeps the master pointer updated; you can always find out where the memory block is because you have the handle with which you can look up the current location.

Ken gave some code for using handles to find memory, and indeed to find other data in the Master Pointer area. Of course code like this has many more applications, as it is just basically a matter of picking up a 32-bit value at a known address, or at an offset from that known address.

How did you guess? I also have written some similar routines! My code is a teensy bit shorter than Ken's, and has the additional advantage of not using any page-zero memory.

I followed the same ground rules as Ken: I assume you are in full 16-bit Native mode (m=x=0), and that the handle address is in the A- and X-registers. The low 16-bits of the handle are in the X-register, and the high 16 are in the A-register. (Of course, addresses in the IIgs are really only 24-bits long, so the high half of the A-register is ignored in the following code.) The result, the 32-bits accessed via the handle, are returned in the A- and X-registers. I wrote two versions, one for inline use, the other a general purpose subroutine.

My first version could be written as a macro, allowing any two pairs of bytes to be picked up in A and X:

```
        .MA PICKUP
        PHB             Save Data Bank Reg
        PHA             Push hi-A, then lo-A
        PLB             Get bank where handle points
        TXY             Get rest of handle in Y-reg
        LDX >]1,Y       Get pair of bytes
        LDA >]2,Y       Get another pair of bytes
        PLB             Pop of what was hi-A
        PLB             Restore B-register
        .EM
```

Use with >PICKUP 0,2 to get first four bytes, the first of these being at the address in the handle. >PICKUP 8,10 will get four bytes starting at 8th. >PICKUP 2,8 will get bytes 2 and 3 in X, 8 and 9 in A.

In the listing which follows, lines 1140-1210 are the same as the macro code above. The program demonstrates using it by printing out the address contained in the four bytes pointed to by a particular handle. The handle in my example contains the address $E10001, so the three bytes beginning at $E10001 are printed out.

Lines 1370-1620 are similar in function, but written as a general subroutine. You can call the subroutine at HANDPTR to get the first four bytes the handle points to, or you can set Y to any offset value and call the subroutine at HANDPTR2 to get an offset group of four bytes.

If you intend to use this subroutine in a larger program that occupies more than one bank, you might want to change the RTS in line 1610 to an RTL, and call the subroutine with a JSL rather than a JSR instruction.

I toyed with the idea of a similar subroutine written in purely 6502 code. What if we called a subroutine with the hi-half of a 16-bit address in the A-register, and the lo-half in the X-register? What code would it take to pickup a two-byte value at an offset from that address? Here is what I came up with, using two bytes of page zero memory:

```
HANDPTR   LDY #0
HANDPTR2  STX ZP
          STA ZP+1
          LDA (ZP),Y
          TAX
          INY
          LDA (ZP),Y
          RTS
```

The only other pure 6502 routine I thought of involved self-modifying code, storing the address inside two LDA instructions.

```
                      1010        .OP 65816
                      1020 *--------------------------------
                      1030 T
000800- 18            1040        CLC
000801- FB            1050        XCE
000802- C2 30         1060        REP #$30
                      1070 *--------------------------------
000804- AD 33 08      1080        LDA HANDLE+2
000807- AE 31 08      1090        LDX HANDLE
                      1100 *--------------------------------
                      1110 *   Standard code sequence to get pointer into A,X
                      1120 *      from a handle in A,X -- 12 bytes.
                      1130 *--------------------------------
00080A- 8B            1140        PHB          Save Data Bank Register
00080B- 48            1150        PHA          Push hi-A (GARBAGE), then lo-A
00080C- AB            1160        PLB          lo-A is bank where handle points
00080D- 9B            1170        TXY          Use 16-bits of address in Y-register
00080E- BE 00 00      1180        LDX >0,Y     Get first two bytes handle pointed at
000811- B9 02 00      1190        LDA >2,Y     Get next two bytes handle pointed at
000814- AB            1200        PLB          pop original hi-A
000815- AB            1210        PLB          Restore Data Bank Register
                      1220 *--------------------------------
000816- 8D 2F 08      1230        STA POINTER+2
000819- 8E 2D 08      1240        STX POINTER
                      1250 *--------------------------------
```

```
00081C- 38        1260        SEC            Print the 24-bit address returned
00081D- FB        1270        XCE
00081E- 20 DA FD  1280        JSR $FDDA
000821- AD 2E 08  1290        LDA POINTER+1
000824- 20 DA FD  1300        JSR $FDDA
000827- AD 2D 08  1310        LDA POINTER
00082A- 4C DA FD  1320        JMP $FDDA
                  1330 *--------------------------------
00082D-           1340 POINTER    .BS 4
000831- 01 00 E1  1350 HANDLE     .DA <$E10001
                  1360 *--------------------------------
                  1370 *   More general subroutine for getting four bytes of data
                  1380 *       from a block of memory pointed at by address in A,X
                  1390 *       ( 18 bytes )
                  1400 *
                  1410 *   Use JSR HANDPTR to get first four bytes.
                  1420 *   Use LDY ##n and JSR HANDPTR2
                  1430 *       to get four bytes starting at nth byte.
                  1440 *   <<Note this subroutine assumes full 16-bit mode>>
                  1450 *--------------------------------
                  1460 HANDPTR
000834- A0 00 00  1470        LDY ##0000
                  1480 HANDPTR2
000837- 8B        1490        PHB            Save Data Bank Register
000838- 48        1500        PHA            Push hi-A (GARBAGE), then lo-A
000839- AB        1510        PLB            lo-A is bank where handle points
00083A- DA        1520        PHX            Push 16-bit address of handle on stack
00083B- B3 01     1530        LDA (1,S),Y    Get 2 bytes at (handle),Y
00083D- AA        1540        TAX            ...save 'em
00083E- C8        1550        INY            Point to next two bytes
00083F- C8        1560        INY
000840- B3 01     1570        LDA (1,S),Y    Get 2 bytes following the other two
000842- 7A        1580        PLY            Pop the handle address
000843- AB        1590        PLB            pop original hi-A
000844- AB        1600        PLB            Restore Data Bank Register
000845- 60        1610        RTS
                  1620 *--------------------------------
```

Converting BCD to Binary, Packing Fields...Bob Sander-Cederlof

I have been working on some hardware recently which includes a date and time chip. The chip produces the year, month, day, hour, minute, and second as six BCD values. That is, each value is coded as an 8-bit byte, but not in binary. The first four bits are the ten's digit of the decimal value, and the other four bits are the unit's digit. This is called BCD, for Binary-Coded-Decimal.

This is nice for display purposes, but not so nice for packing into a binary format. My operating system needs the date and time packed into four bytes. (ProDOS does it in much the same way.) The end result will be two 16-bit values, looking like this:

```
        YYYYYYYMMMMDDDDD
        hhhhhmmmmmmsssss
```

YYYYYYY means a seven bit field for the year, with a value between 0 and 99; MMMM is the month, 1-12; DDDDD is the day of month, 1-31; hhhhh is the hour of the day, 0-23; mmmmmm is the minute, 0-59; and sssss is for seconds, but only runs from 0 to 29. There are not quite enough bits, so "sssss" is equal to seconds/2. This just happens to be the way date and time are stored in MS/DOS file directory entries, by the way.

To start with, I needed an efficient way to convert a BCD byte into a binary value. Since I was working on a 65816-based system, I coded with that processor in mind. The listing which follows shows three different versions of this subroutine. The third one is written to run in a plain-vanilla 6502, in case that is all you have.

The first version, lines 1020-1210, takes 20 bytes. It uses the stack for temporary storage, and works by isolating the ten's digit, calculating the binary value of ten times the ten's digit, and adding the unit's digit. I used the Stack-Relative addressing mode here, so it does require the 65816 or 65802 processor. It will work in either Native or Emulation mode. If you are in Native mode, the m-bit must be 1 so the A-register works as an 8-bit register.

The second version, lines 1220-1400, is only 18 bytes. I got a little trickier, and took advantage of the fact that 10x is equal to 16x-6x. This also uses the Stack-Relative address mode, so the same restrictions apply as with the first version.

The third version, lines 1410-1590, which will run in a 6502 or 65C02, takes 22 bytes as shown. It requires two bytes for temporary storage. (I include these two bytes in the count.) If you put the two temp bytes in page zero, it will shorten the code by four bytes (still counting the temp bytes) making it just as short as the shortest 65816 version! Another shortening option would require the subroutine to be in RAM: change lines 1530 and 1540 to use immediate mode, and store the T1 and T2 values directly into the address fields of these two instructions. This would also make an 18-byte subroutine, but with the stigma of being self-modifying code.

I wrote a test routine, to be sure my subroutines worked correctly. Lines 1630-1800 run through all 100 possible values, comparing the converted result with the expected result. If there are any discrepancies, I print out the BCD and Binary values. Naturally, they all worked perfectly and I got no printout. (When that happens it is a good idea to purposely insert a bug in the subroutine being tested to make sure the test routine itself is working!)

The test routine uses a STZ opcode, which is on the 65C02 and up, but not on the 6502. Substitute LDA #0, STA 0 is you have a 6502. The test routine counts from 0 to 99 in decimal mode in the X- and A-registers, and from 0 to $63 in binary mode in page zero location $00.

Lines 1810-2180 call on one of the BCD-to-BIN converters to convert the date and time values, and then use 6502-compatible code to pack it all into the required four-byte format. I used a sample date and time in lines 2200-2280.

```
                        1000        .OP 65816
                        SAVE S.CONV.BCD.TO.BIN
                        1020 *--------------------------------
                        1030 *   Convert BCD to BIN by parts
                        1040 *--------------------------------
                        1050 CONV.BCD.TO.BIN.1
000800- 48              1060        PHA
000801- 29 0F           1070        AND #$0F      ISOLATE UNITS DIGIT
000803- 48              1080        PHA
000804- 43 02           1090        EOR 2,S       ISOLATE TENS DIGIT
000806- 4A              1100        LSR           TENS*8
000807- 48              1110        PHA
000808- 4A              1120        LSR
000809- 4A              1130        LSR           TENS*2
00080A- 63 01           1140        ADC 1,S       TENS*10
00080C- 63 02           1150        ADC 2,S       TENS*10+UNITS
00080E- 83 03           1160        STA 3,S       save converted value
000810- 68              1170        PLA           POP off temps
000811- 68              1180        PLA
000812- 68              1190        PLA           get converted result
000813- 60              1200        RTS           RETURN
14-                     1210 Z.A    .EQ *-CONV.BCD.TO.BIN.1
```

```
                  1220 *--------------------------------
                  1230 *   Convert BCD to BIN by subtraction
                  1240 *       10a+b = 16a+b - 6a
                  1250 *--------------------------------
                  1260 CONV.BCD.TO.BIN.2
000814- 48        1270        PHA          Save 16*a+b
000815- 29 F0     1280        AND #$F0     Isolate 16*a
000817- 4A        1290        LSR          make it 8*a
000818- 4A        1300        LSR          make it 4*a
000819- 48        1310        PHA          Save 4*a
00081A- 4A        1320        LSR          make 2*a
00081B- 63 01     1330        ADC 1,S      4a+2a = 6a
00081D- E3 02     1340        SBC 2,S      6a - (16a+b) - 1 (because carry was clear)
00081F- 49 FF     1350        EOR #$FF     (16a+b) - 6a
000821- 83 02     1360        STA 2,S      Save in stack
000823- 68        1370        PLA          pop off temp value
000824- 68        1380        PLA          Get binary result
000825- 60        1390        RTS          RETURN
12-               1400 Z.B    .EQ *-CONV.BCD.TO.BIN.2
                  1410 *--------------------------------
                  1420 *   6502 Version
                  1430 *   Convert BCD to BIN by subtraction
                  1440 *       10a+b = 16a+b - 6a
                  1450 *--------------------------------
                  1460 CONV.BCD.TO.BIN.3
000826- 8D 3A 08  1470        STA T1       Save 16*a+b
000829- 29 F0     1480        AND #$F0     Isolate 16*a
00082B- 4A        1490        LSR          make it 8*a
00082C- 4A        1500        LSR          make it 4*a
00082D- 8D 3B 08  1510        STA T2       Save 4*a
000830- 4A        1520        LSR          make 2*a
000831- 6D 3B 08  1530        ADC T2       4a+2a = 6a
000834- ED 3A 08  1540        SBC T1       6a - (16a+b) - 1 (because carry was clear)
000837- 49 FF     1550        EOR #$FF     (16a+b) - 6a
000839- 60        1560        RTS          RETURN
00083A-           1570 T1     .BS 1
00083B-           1580 T2     .BS 1
16-               1590 Z.C    .EQ *-CONV.BCD.TO.BIN.3
                  1600 *--------------------------------
                  1610 *   Test Conversion Subroutine
                  1620 *--------------------------------
00083C- 64 00     1630 U      STZ 0
00083E- A2 00     1640        LDX #0
000840- 8A        1650 .1     TXA
000841- 20 00 08  1660        JSR CONV.BCD.TO.BIN.1
000844- C5 00     1670        CMP 0
000846- F0 07     1680        BEQ .2
000848- 20 DA FD  1690        JSR $FDDA
00084B- 8A        1700        TXA
00084C- 20 DA FD  1710        JSR $FDDA
00084F- 8A        1720 .2     TXA
000850- E6 00     1730        INC 0
000852- F8        1740        SED
000853- 18        1750        CLC
000854- 69 01     1760        ADC #1
000856- D8        1770        CLD
000857- AA        1780        TAX
000858- D0 E6     1790        BNE .1
00085A- 60        1800        RTS
```

```
                        *--------------------------------
                        *   Convert BCD Date/Time to Packed Binary
                        *--------------------------------
                        S
00085B- A2 05                  LDX #5
00085D- BD 9D 08        .1     LDA BCD.DATE.AND.TIME,X
000860- 20 14 08               JSR CONV.BCD.TO.BIN.2
000863- 9D A3 08               STA BIN.DATE.AND.TIME,X
000866- CA                     DEX
000867- 10 F4                  BPL .1
                        *---Pack converted time----------
000869- 4E A8 08               LSR SEC         000SSSSS  Sec/2
00086C- AD A7 08               LDA MIN         00MMMMMM
00086F- 0A                     ASL             0MMMMMM0
000870- 0A                     ASL             MMMMMM00
000871- 0A                     ASL             M.MMMMM000
000872- 2E A6 08               ROL HOUR        00HHHHHM
000875- 0A                     ASL             M.MMMM0000
000876- 2E A6 08               ROL HOUR        0HHHHHMM
000879- 0A                     ASL             M.MMM00000
00087A- 0D A8 08               ORA SEC         MMMSSSSS
00087D- 8D AB 08               STA HMS
000880- AD A6 08               LDA HOUR
000883- 2A                     ROL             HHHHHMMM
000884- 8D AC 08               STA HMS+1       HHHHHMMM
                        *---Pack converted date----------
000887- AD A4 08               LDA MONTH       0000mmmm
00088A- 0A                     ASL             000mmmm0
00088B- 0A                     ASL             00mmmm00
00088C- 0A                     ASL             0mmmm000
00088D- 0A                     ASL             mmmm0000
00088E- 0A                     ASL             m.mmm00000
00088F- 0D A5 08               ORA DAY         mmmddddd
000892- 8D A9 08               STA YMD
000895- AD A3 08               LDA YEAR        0yyyyyyy
000898- 2A                     ROL             yyyyyyym
000899- 8D AA 08               STA YMD+1
00089C- 60                     RTS
                        *--------------------------------
                        *   Date and Time in BCD Format
                        *--------------------------------
                        BCD.DATE.AND.TIME
00089D- 87                     .HS 87          Year
00089E- 12                     .HS 12          Month
00089F- 17                     .HS 17          Day
0008A0- 09                     .HS 09          Hour
0008A1- 57                     .HS 57          Minute
0008A2- 30                     .HS 30          Second
                        *--------------------------------
                        BIN.DATE.AND.TIME
0008A3-                 YEAR   .BS 1           TEMPS, receive binary values
0008A4-                 MONTH  .BS 1
0008A5-                 DAY    .BS 1
0008A6-                 HOUR   .BS 1
0008A7-                 MIN    .BS 1
0008A8-                 SEC    .BS 1
                        *--------------------------------
                        *   Date and Time in Packed Binary Format
                        *--------------------------------
0008A9-                 YMD    .BS 2           YYYYYYY.MMMM.DDDDD
0008AB-                 HMS    .BS 2           HHHHH.MMMMMM.SSSSS  SSSSS=Sec/2
                        *--------------------------------
```

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are $1.80 each (other countries inquire for postage). A subscription to the newsletter and the Monthly Disk containing all source code is $64 per year in the US, Canada and Mexico, and $87 to other countries.